

Python: module vcs.utils

vcs.utils

[index](#)

Modules

[cdtime](#)

Classes

[exceptions.Exception](#)
[VCSUtilsError](#)

class **VCSUtilsError**([exceptions.Exception](#))

Methods defined here:

__init__(self, args=None)
 Create an exception

__repr__ = **__str__**(self)

__str__(self)
 Calculate the string representation

Methods inherited from [exceptions.Exception](#):

__getitem__(...)

Functions

generate_time_labels(d1, d2, units, calendar=135441)

[generate_time_labels](#)(self, d1, d2, units, calendar=[cdtime.DefaultCalendar](#))
returns a dictionary of time labels for an interval of time, in a
d1 and d2 must be [cdtime](#) object, if not they will be assumed to be

Example:

```
lbls = generate\_time\_labels(cdtime.reltime\(0, 'months since 2000'\),  
                         cdtime.reltime\(12, 'months since 2000'\),  
                          'a days since 1800',  
                         )
```

This generated a dictionary of nice time labels for the year 2000

```
lbls = generate_time_labels(cdtimes.reltimes(0,'months since 2000'),  
                           cdtimes.comptime(2001),  
                           'days since 1800',  
                           )
```

This generated a dictionary of nice time labels for the year 2000

```
lbls = generate_time_labels(0,  
                           12,  
                           'months since 2000',  
                           )
```

This generated a dictionary of nice time labels for the year 2000

getcolors(levs, colors=[16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, ...], split=1, white=240)
Function : getcolors(levs, colors=range(16,240), split=1, white=240)

Description of Function:

For isofill/boxfill purposes

Given a list of levels this function returns the colors that would

If the colors are split an interval goes from <0 to >0 then this
Usage:

levs : levels defining the color ranges

colors (default= range(16,240)) : A list/tuple of the colors

split # parameter to split the colors between 2 equal domain:

one for positive values and one for negative values

0 : no split

1 : split if the levels go from <0 to >0

2 : split even if all the values are positive or negative

white (=240) # If split is on and an interval goes from <0 to >0

Examples of Use:

```
>>> a=[0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0]  
>>> vcs.getcolors (a)  
[16, 41, 66, 90, 115, 140, 165, 189, 214, 239]  
>>> vcs.getcolors (a,colors=range(16,200))  
[16, 36, 57, 77, 97, 118, 138, 158, 179, 199]  
>>> vcs.getcolors(a,colors=[16,25,15,56,35,234,12,11,19,32,132,141])  
[16, 25, 15, 35, 234, 12, 11, 32, 132, 17]  
>>> a=[-6.0, -2.0, 2.0, 6.0, 10.0, 14.0, 18.0, 22.0, 26.0]  
>>> vcs.getcolors (a,white=241)  
[72, 241, 128, 150, 172, 195, 217, 239]  
>>> vcs.getcolors (a,white=241,split=0)  
[16, 48, 80, 112, 143, 175, 207, 239]
```

minmax(*data)

Function : minmax

Description of Function

Return the minimum and maximum of a serie of array/list/tuples (

Values those absolute value are greater than 1.E20, are masked

You can combined list/tuples/... pretty much any combination is

Examples of Use

```
>>> s=range(7)
>>> vcs.minmax(s)
(0.0, 6.0)
>>> vcs.minmax([s,s])
(0.0, 6.0)
>>> vcs.minmax([[s,s*2],4.,[6.,7.,s]],[5.,-7.,8,(6.,1.)])
(-7.0, 8.0)
```

mkevenlevels(n1, n2, nlev=10)

Function : mkevenlevels

Description of Function:

Return a serie of evenly spaced levels going from n1 to n2
by default 10 intervals will be produced

Examples of use:

```
>>> vcs.mkevenlevels(0,100)
[0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
>>> vcs.mkevenlevels(0,100,nlev=5)
[0.0, 20.0, 40.0, 60.0, 80.0, 100.0]
>>> vcs.mkevenlevels(100,0,nlev=5)
[100.0, 80.0, 60.0, 40.0, 20.0, 0.0]
```

mklabels(vals, output='dict')

Function : mklabels

Description of Function:

This function gets levels and output strings for nice display of

Examples of use:

```
>>> a=vcs.mkscale(2,20,zero=2)
>>> vcs.mklabels (a)
{20.0: '20', 18.0: '18', 16.0: '16', 14.0: '14', 12.0: '12', 10.0: '10', 8.0: '8', 6.0: '6', 4.0: '4', 2.0: '2', 0.0: '0'}
>>> vcs.mklabels ( [5,.005])
{0.005000000000000001: '0.005', 5.0: '5.000'}
>>> vcs.mklabels ( [.00002,.00005])
{2.0000000000000002e-05: '2E-5', 5.0000000000000002e-05: '5E-5'}
>>> vcs.mklabels ( [.00002,.00005],output='list')
['2E-5', '5E-5']
```

mkscale(n1, n2, nc=12, zero=1)

Function: mkscale

Description of function:

This function return a nice scale given a min and a max

option:

```
nc # Maximum number of intervals (default=12)
zero # Not all implemented yet so set to 1 but values will be:
      -1: zero MUST NOT be a contour
      0: let the function decide # NOT IMPLEMENTED
```

```
1: zero CAN be a contour (default)
2: zero MUST be a contour
Examples of Use:
>>> vcs.mkscale(0,100)
[0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
>>> vcs.mkscale(0,100,nc=5)
[0.0, 20.0, 40.0, 60.0, 80.0, 100.0]
>>> vcs.mkscale(-10,100,nc=5)
[-25.0, 0.0, 25.0, 50.0, 75.0, 100.0]
>>> vcs.mkscale(-10,100,nc=5,zero=-1)
[-20.0, 20.0, 60.0, 100.0]
>>> vcs.mkscale(2,20)
[2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0]
>>> vcs.mkscale(2,20,zero=2)
[0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0]
```